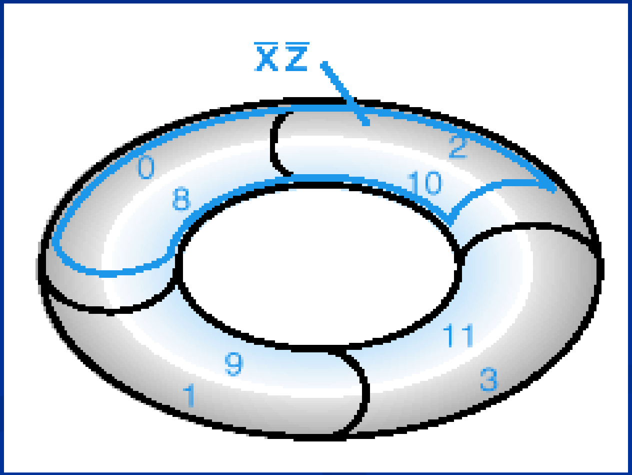
Lecture 3 : Gate-Level Minimization

Outline

- four-variable Karnaugh Map
- NAND and NOR Implementations
- Other Two-Level Implementations
- Exclusive-OR Function

Four-Variable Maps

WX \ YZ	00	01	11	10
00	m_0	m_1	m_3	m_2
01	m_4	m_5	m_7	m_6
11	m_{12}	m_{13}	m_{15}	m_{14}
10	m_8	m_9	m_{11}	m_{10}



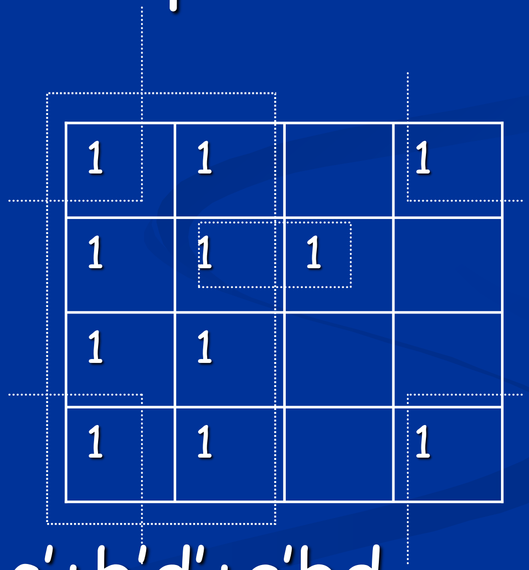
- Top cells are adjacent to bottom cells. Left-edge cells are adjacent to right-edge cells.
- Note variable ordering (WXYZ).

Example

- Simplify the following Boolean function (A,B,C,D) = ∑m(0,1,2,4,5,7,8,9,10,12,13).
- First put the function g( ) into the map, and then group as many 1s as possible.

ab \ cd

1	1		1
1	1	1	
1	1		
1	1		1

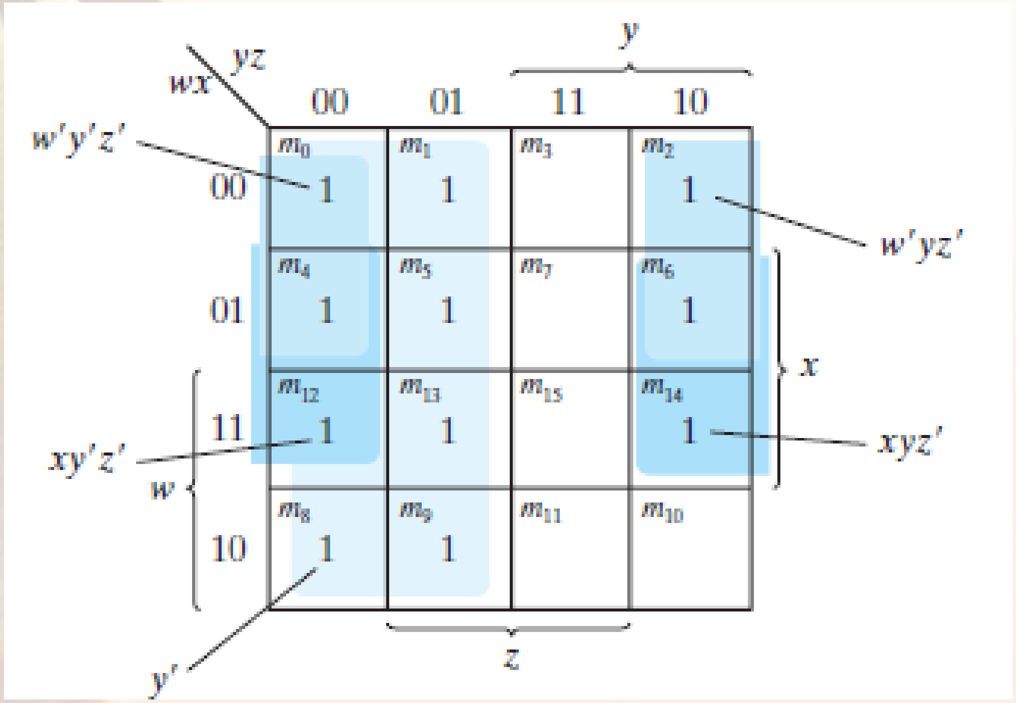


g(A,B,C,D) = c′+b′d′+a′bd

EXAMPLE 3.5

Simplify the Boolean function

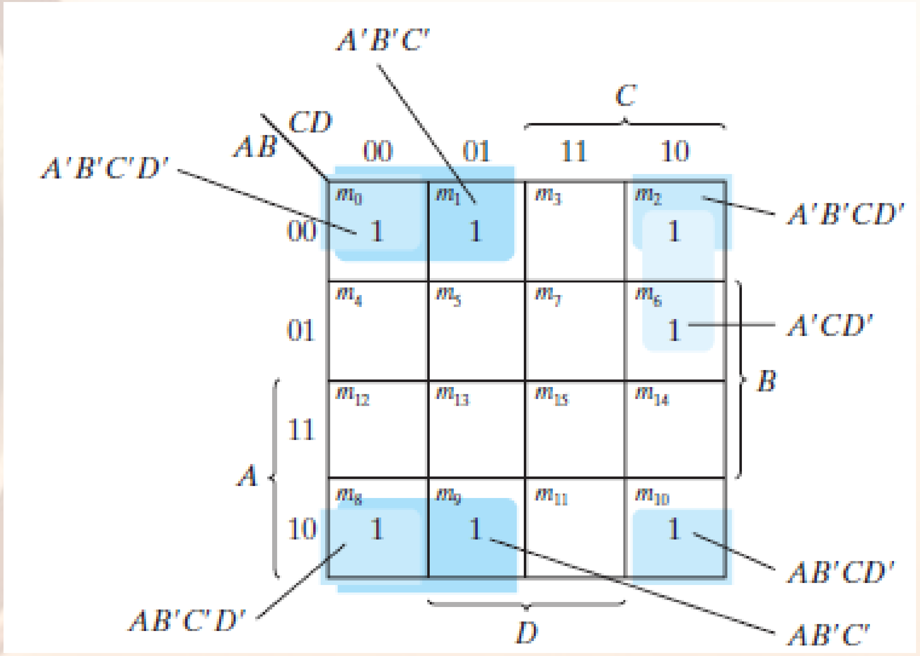
F (w, x, y, z) =Σ (0, 1, 2, 4, 5, 6, 8, 9, 12, 13, 14)



$$F = y' + w'z' + xz'$$

EXAMPLE 3.6
Simplify the Boolean function

$$F = A'B'C' + B'CD' + A'BCD' + AB'C'$$



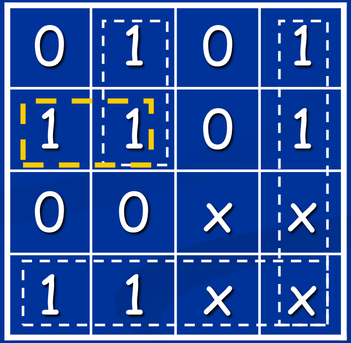
$$F = B'D' + B'C' + A'CD'$$

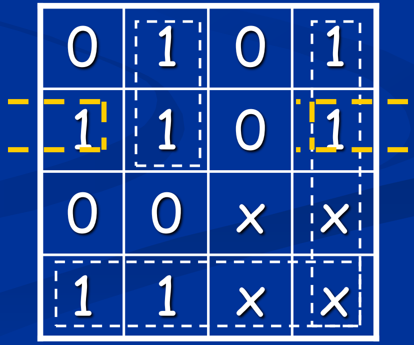
Example

ab \ cd	00	01	11	10
00	0	1	0	1
01	1	1	0	1
11	0	0	x	x
10	1	1	x	x

- Simplify the function f(a,b,c,d) whose K-map is shown at the right.
- f = a'c'd+ab'+cd'+a'bc'

  or
- f = a'c'd+ab'+cd'+a'bd'
- The middle two terms are EPIs, while the first and last terms are selected to
  cover the minterms m_1, m_4, and m_5.
- (There's a third solution!)

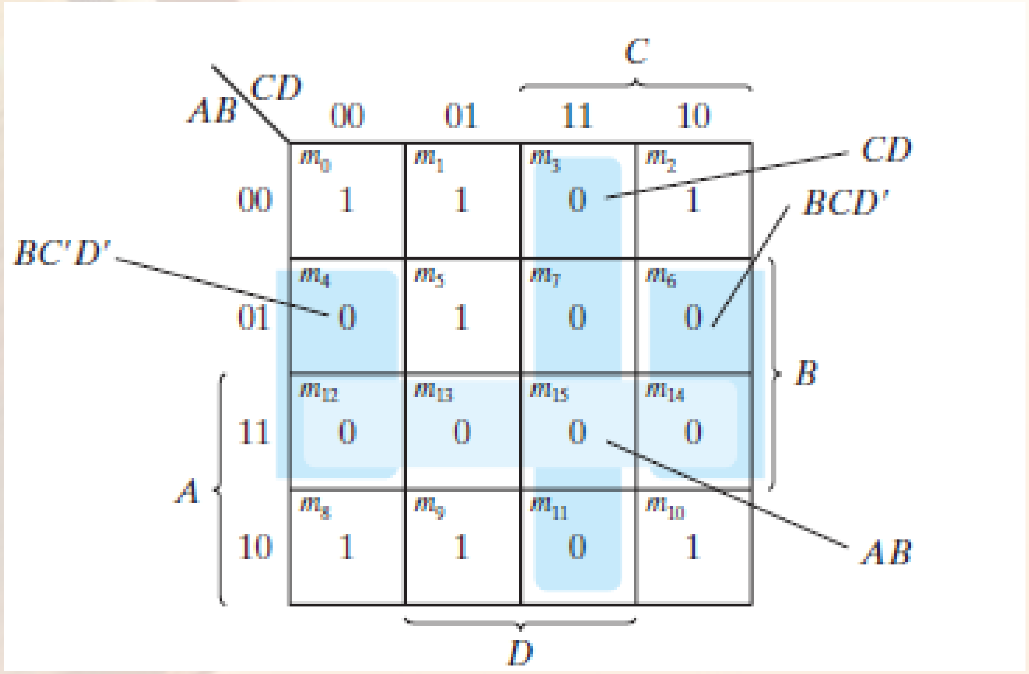




EXAMPLE 3.7

Simplify the following Boolean function into (a) sum-of-products form and (b) product-of-sums form:

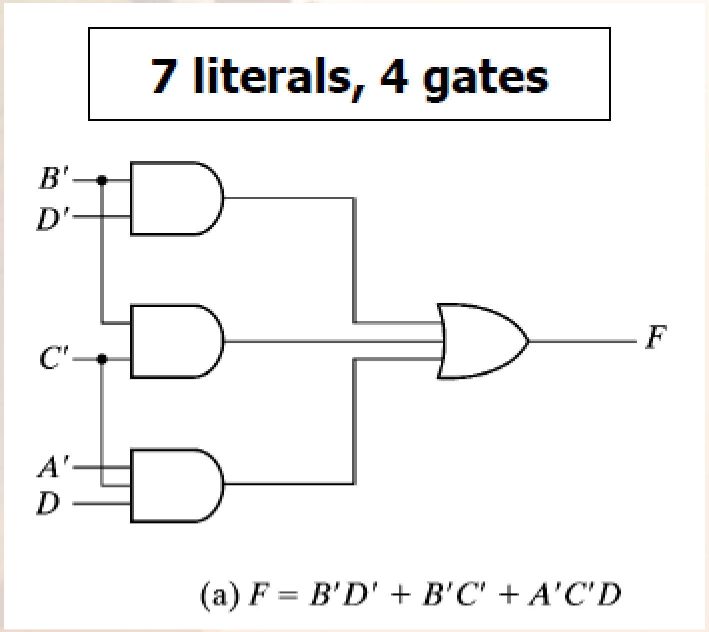
F (A, B, C, D) =Σ(0, 1, 2, 5, 8, 9, 10)



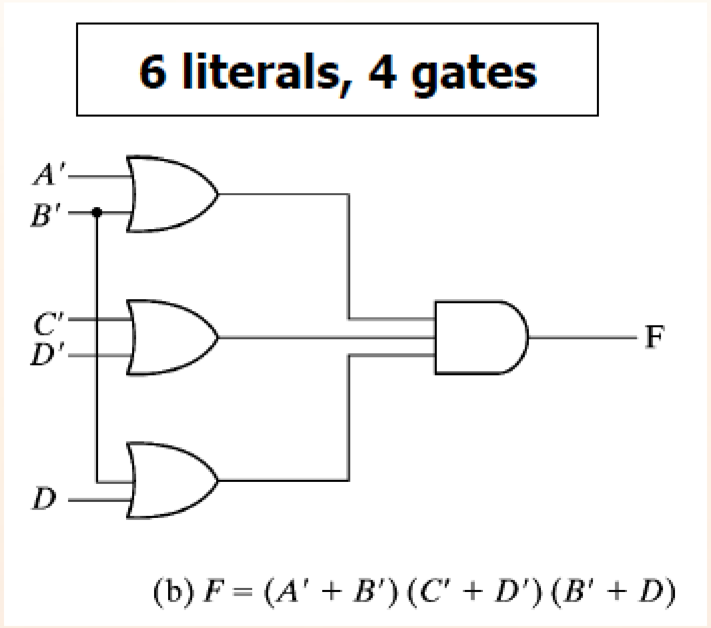
$$\text{(b)}\quad F = (A' + B')(C' + D')(B' + D)$$

Two Gate Implementations

- Sometimes product-of-sums representations may have smaller implementations



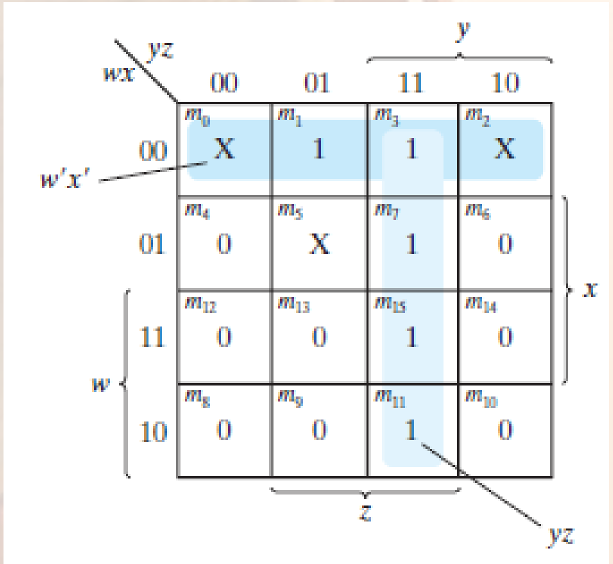
(a) $F = B'D' + B'C' + A'C'D$



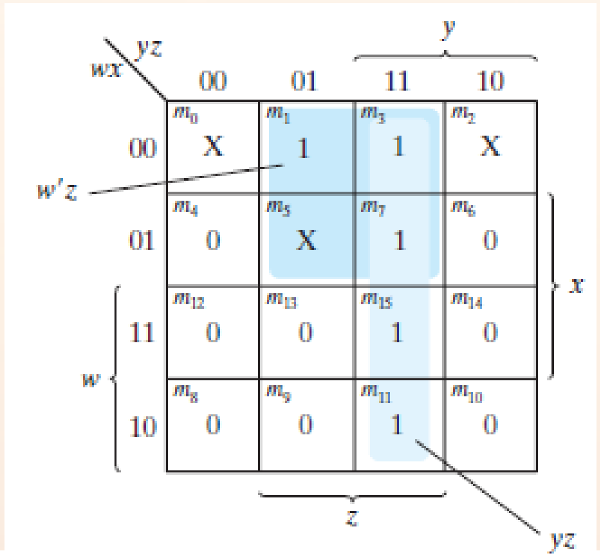
(b) $F = (A' + B')(C' + D')(B' + D)$

Don't Care Conditions

- X = don't care (can be 0 or 1)
- Don't cares can be included to form a larger cube, but not necessary to be completely covered
- Ex: *F(w, x, y, z) =Σ(1,3,7,11,15) d(w, x, y, z) =Σ(0,2,5)*



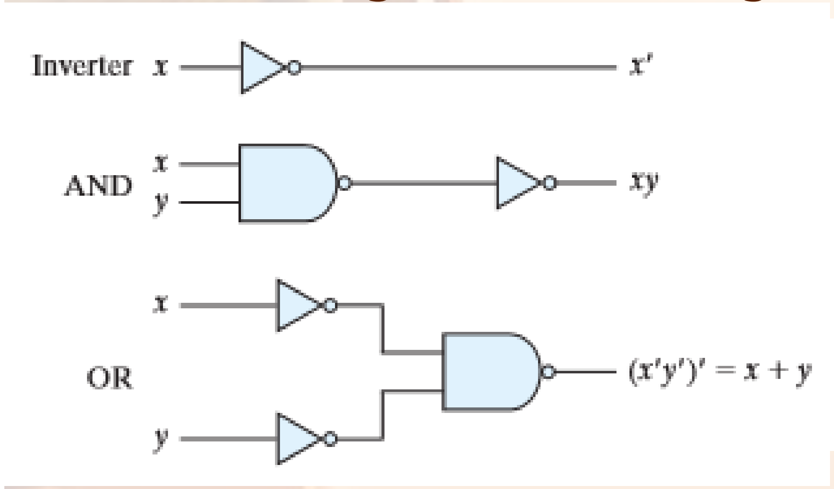
(a) $F = yz + w'x'$

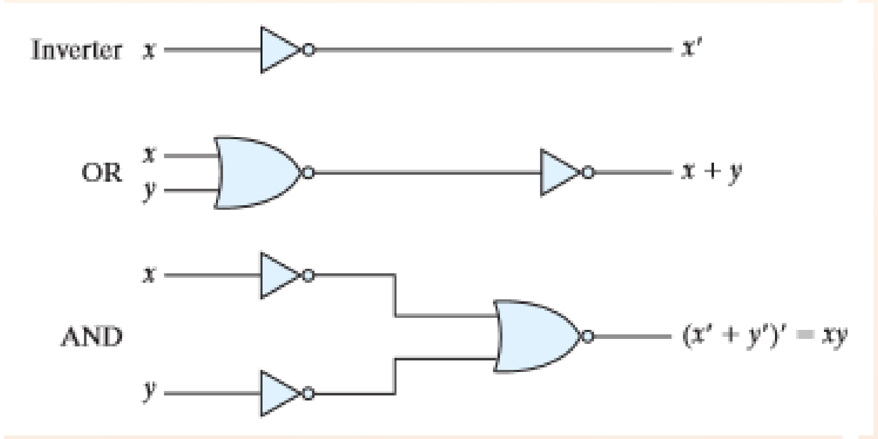


(b) $F = yz + w'z$

NAND and NOR Implementation

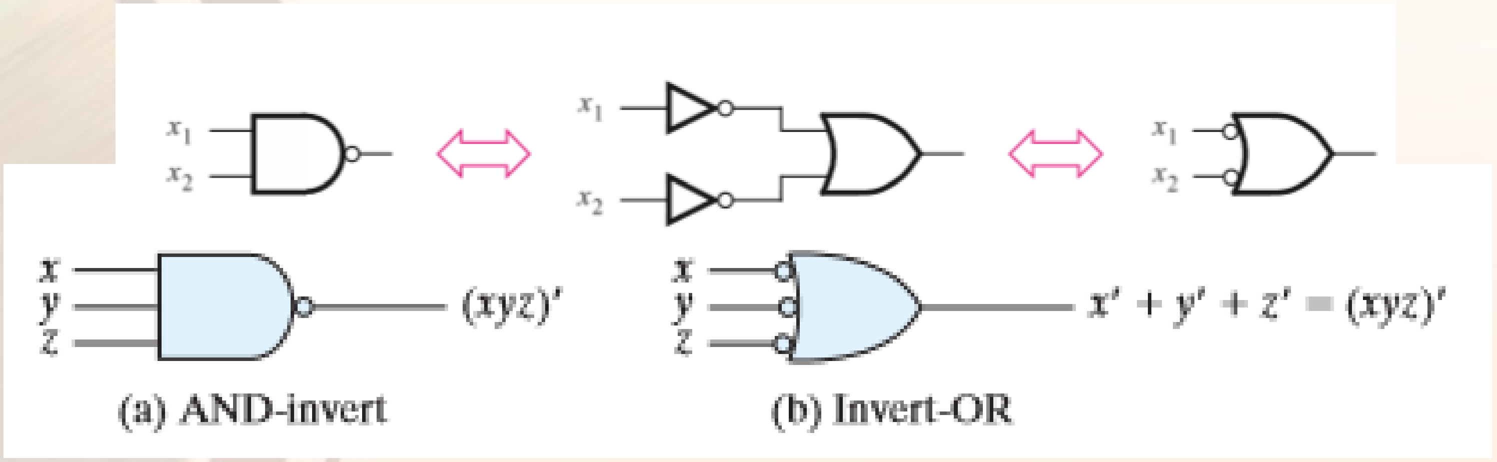
- Digital circuits are frequently constructed with **NAND** or **NOR gates rather than with AND and OR gate**
  - NAND and NOR gates are much easier to fabricate
- NAND or NOR gates are both universal gates
  - Any digital system can be implemented with only NAND gates or NOR gates

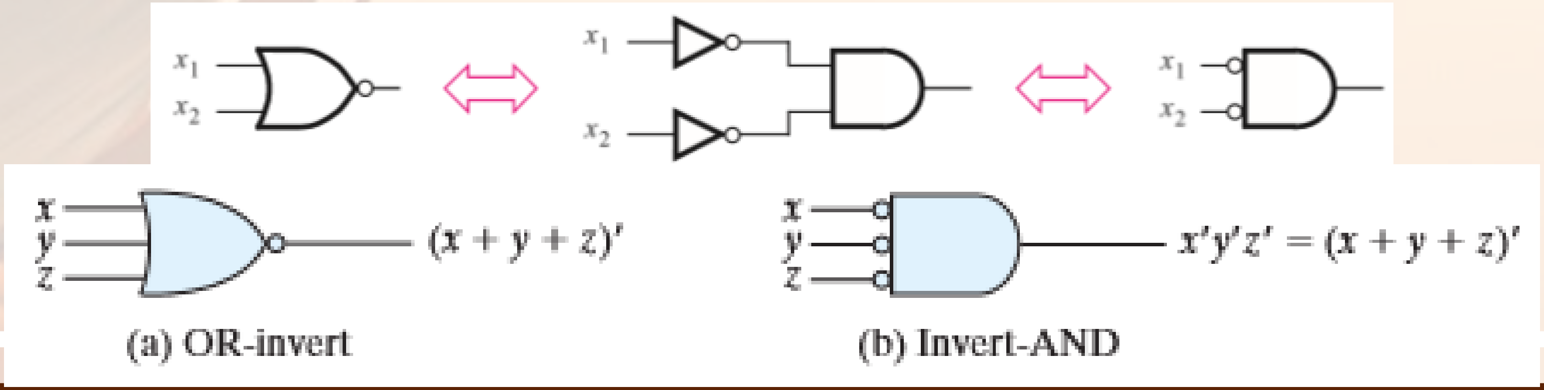




Alternative Graphic Symbols

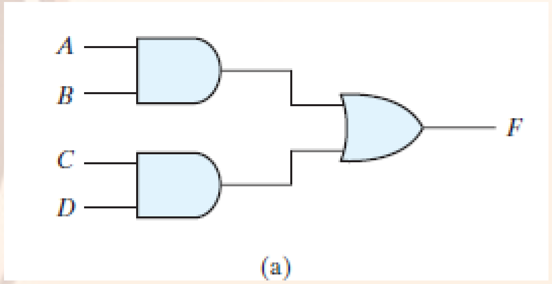
- To facilitate the conversion to NAND or NOR logic, it is convenient to define alternative graphic symbols
  - "Bubble" means complement



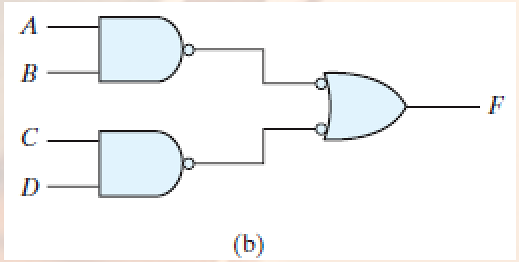


Two-Level Implementation (NAND)

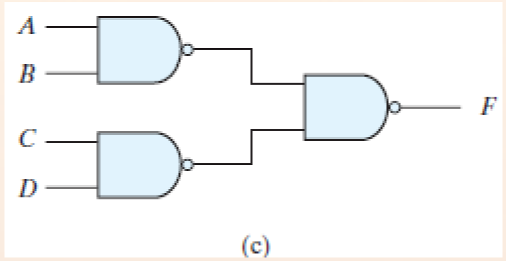
- It's easy to implement a Boolean function with only NAND gates if converted from a **sum of products form**
- Ex: F = AB+CD = ((AB)'(CD)')'



(a)



(b)



(c)

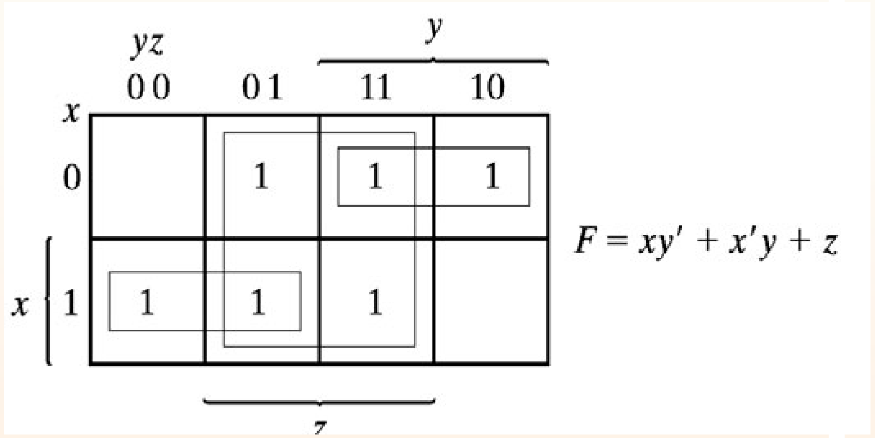
EXAMPLE 3.9

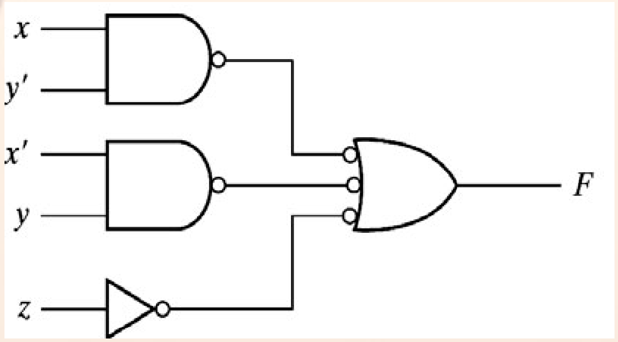
Implement the following Boolean function with NAND gates:

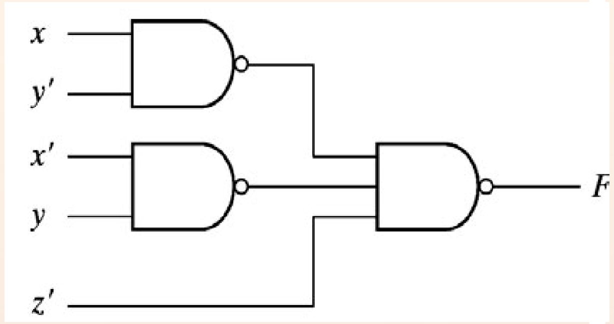
F (x, y, z) = Σ(1, 2, 3, 4, 5, 7)

Procedures:

1. Simplify the function in sum of products
2. Draw NAND gates for the first level
3. Draw a single AND-invert or invert-OR in the second level
4. Add an inverter at the first level for the term with a single literal

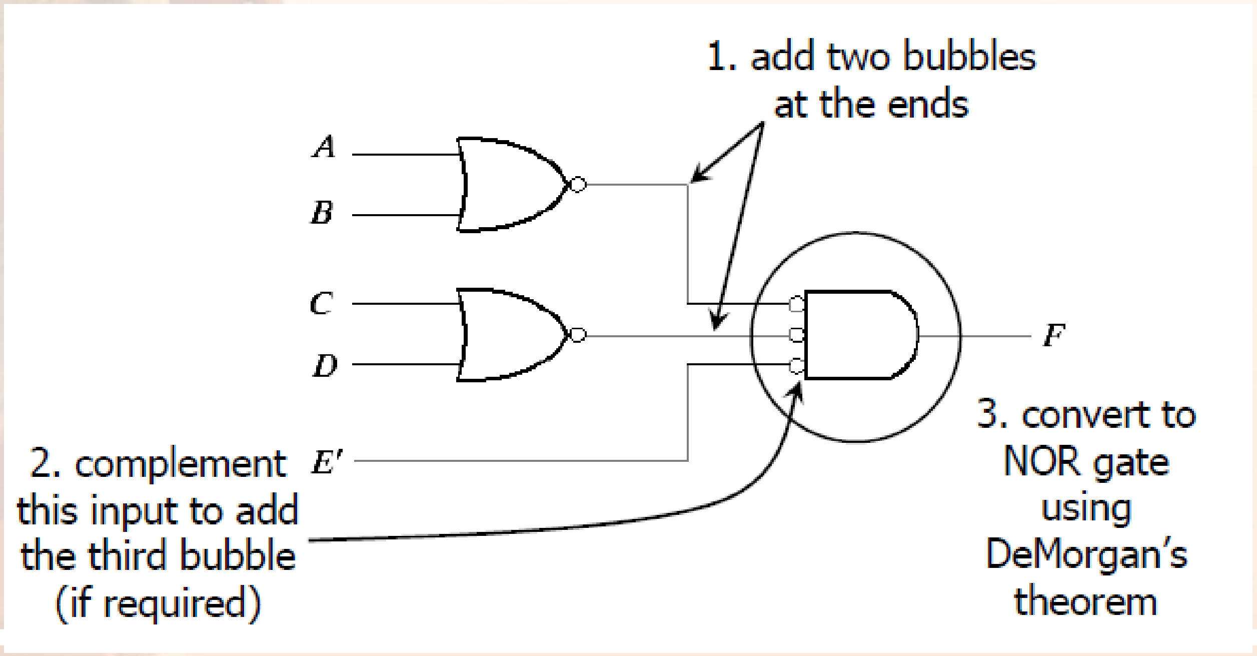






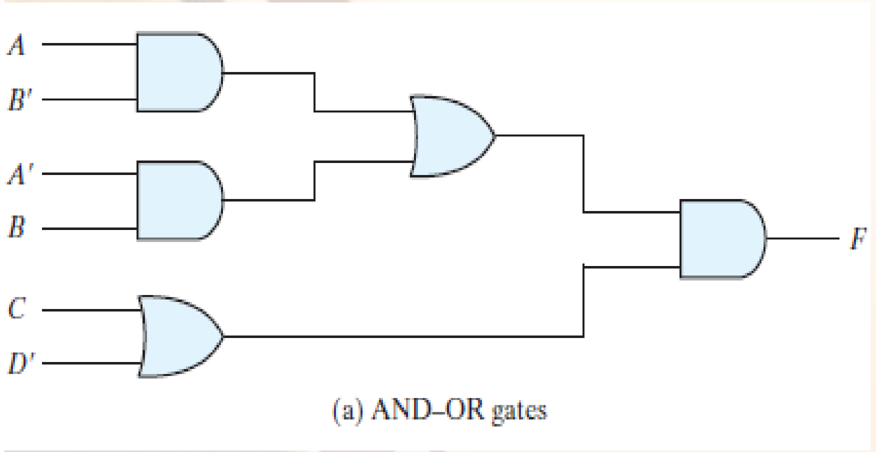
Two-Level Implementation (NOR)

- It's easy to implement a Boolean function with only NOR gates if converted from a **product of sums form**
- Ex: F=(A+B)(C+D)E

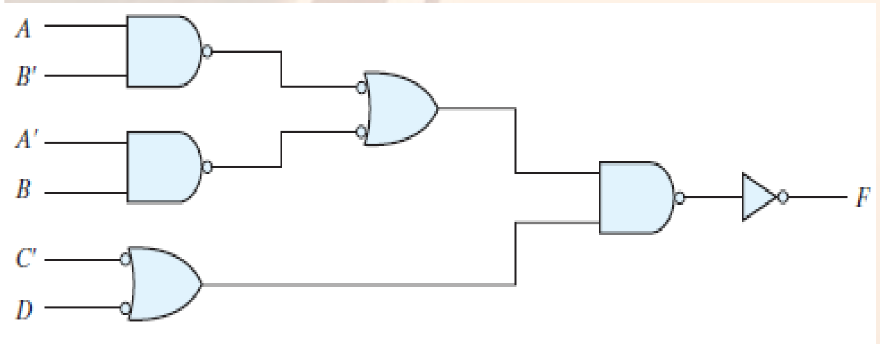

1. add two bubbles at the ends
A
B
C
D
F
2. complement this input to add the third bubble (if required)
E′
3. convert to NOR gate using DeMorgan's theorem

Multilevel NAND Circuits

Implementing ***F = (AB' + A'B) (C + D')***



(a) AND–OR gates



(b) NAND gates

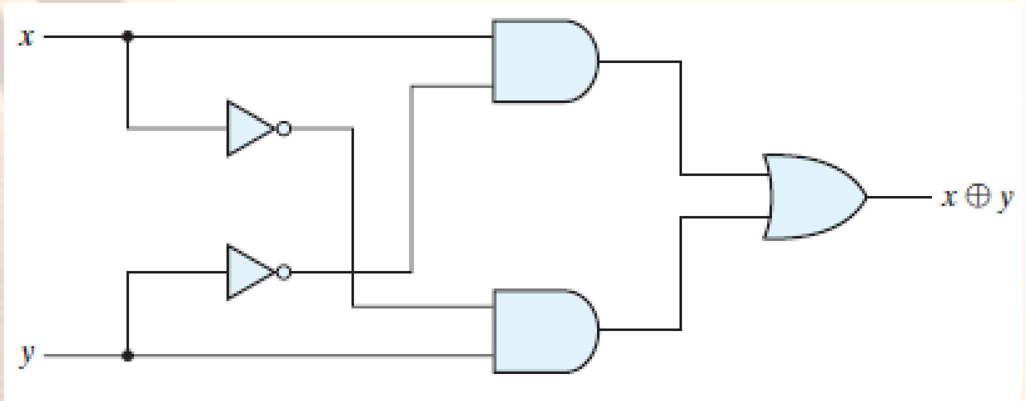
- ❑ Procedures:

1. Convert all AND gates to NAND gates with AND-invert symbols
2. Convert all OR gates to NAND gates with invert-OR symbols
3. Check all bubbles and insert an inverter for the bubble that are not compensated by another bubble

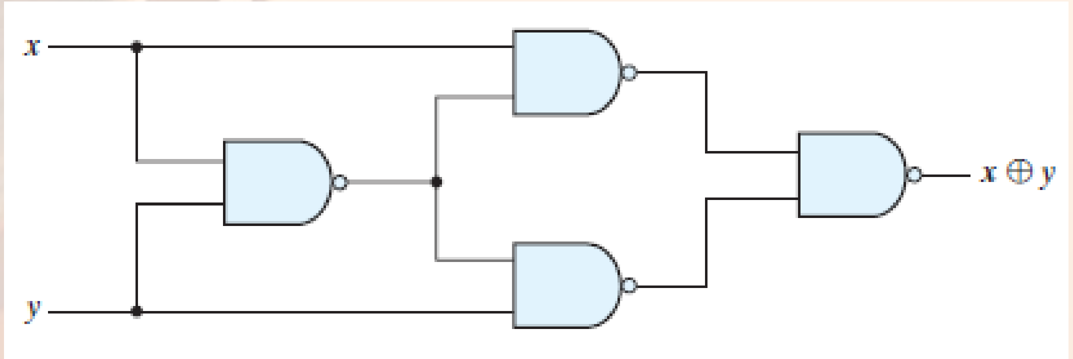
Exclusive-OR (XOR) Function

- XOR is often denoted by the symbol $\oplus$
- Logic operation of XOR
  - $X \oplus Y = XY' + X'Y$
  - Equal to 1 if **only x is equal to 1 or if only y is equal to 1,** but not when both are equal to 1
- It's complement, exclusive-NOR (XNOR), is often denoted by the symbol $\odot$
- Logic operation
  - $X \odot Y = XY + X'Y'$
  - It is equal to 1 if **both x and y are equal to 1 or if both are** equal to 0
- Seldom used in general Boolean functions
  - Particularly useful in arithmetic operations and error detection and correction circuits

Exclusive-OR Implementations



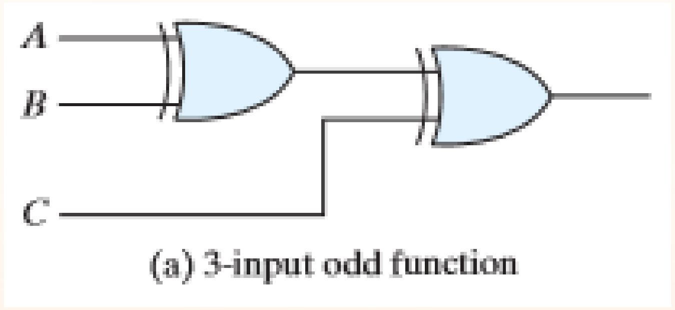
(a) Exclusive-OR with AND–OR–NOT gates



(b) Exclusive-OR with NAND gates

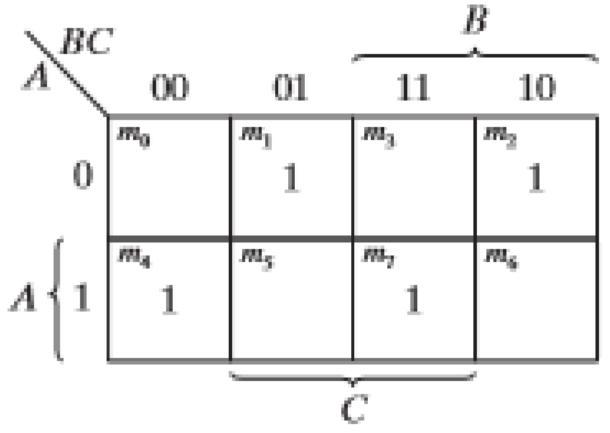
Odd Function

- The multiple-variable XOR operation is defined as an **odd function**
  - TRUE when no. of "1" in inputs is odd



(a) 3-input odd function

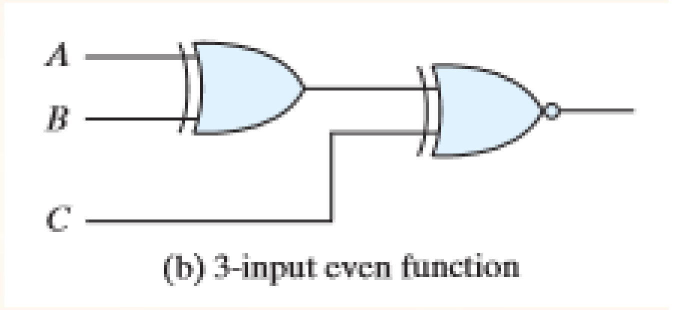
A	B	C	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1



(a) Odd function $F = A \oplus B \oplus C$

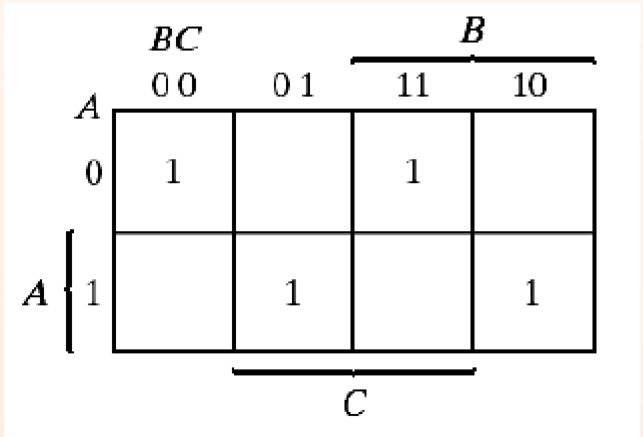
Even Function

- The multiple-variable XNOR operation is defined as an **even function**
  - TRUE when no. of “1” in inputs is even



(b) 3-input even function

A	B	C	F
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0



A \ BC	00	01	11	10
0	1		1	
1		1		1

(a) Even function
$F = (A \oplus B \oplus C)'$

Parity Generation and Checking

- An extra **parity bit is often added and** checked at the receiving end for error
- The circuit that generates the parity bit in the transmitter is called a **parity generator**
- The circuit that checks the parity in the receiver is called a **parity checker**
- Exclusive-OR functions are very useful to construct such circuits

Parity Generator

- For even parity:
- The total number of "1" (including P) is even
- The number of "1" at inputs is odd
- Generated with an **XOR** gate (odd function)
- P = x ⊕ y ⊕ z (for 3-bit message)
- Similarly, odd parity can be generated with an XNOR gate

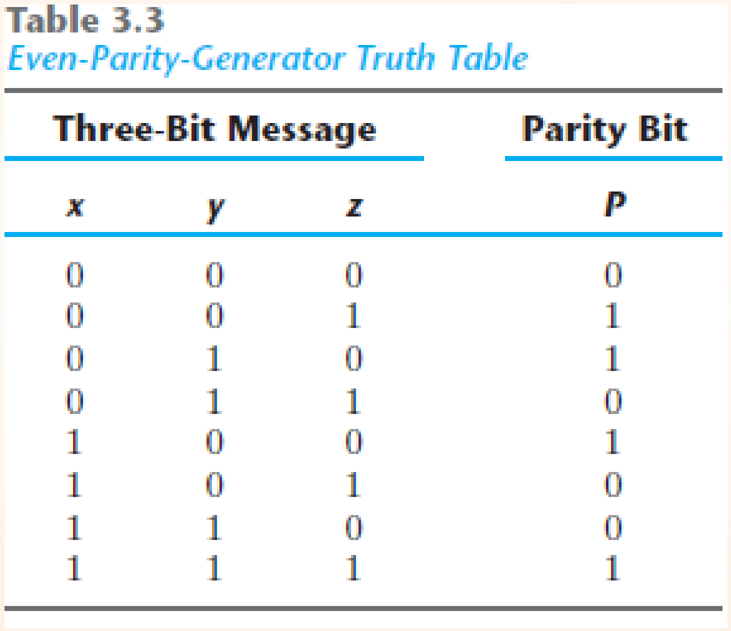
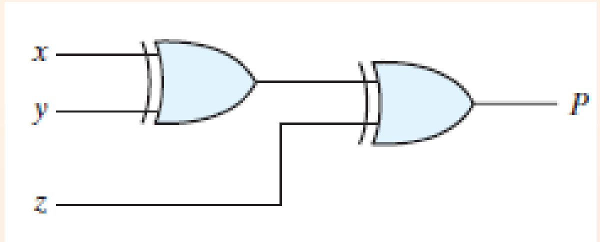


Table 3.3
Even-Parity-Generator Truth Table

Three-Bit Message			Parity Bit
x	*y*	*z*	*P*
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1



(a) 3-bit even parity generator

$$P = x \oplus y \oplus z$$

Parity Checker

- For even parity, the total number of "1" in the message is even
  - An error occurs when the received number of "1" is odd
- An XOR gate (odd function) can detect such an error
  - Has n+1 inputs

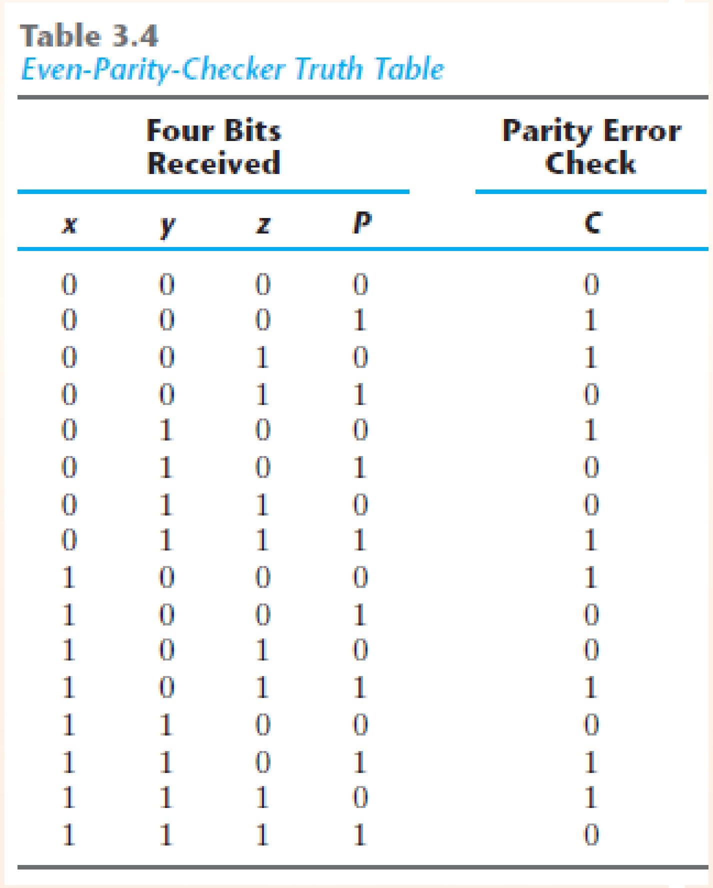
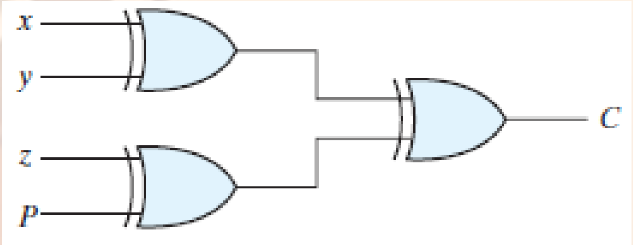


Table 3.4
Even-Parity-Checker Truth Table

Four Bits Received				Parity Error Check
x	**y**	**z**	**P**	**C**
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

$$C = x \oplus y \oplus z \oplus P$$



(b) 4-bit even parity checker